EDK II Standard Libraries ReadMe

Alpha 3 Release

OVERVIEW

This document describes the EDK II specific aspects of installing, building, and using the Standard C Library component of the EDK II Application Development Kit, EADK.

The EADK is comprised of three packages: AppPkg, StdLib, and StdLibPrivateInternalFiles.

AppPkg	This package contains applications which demonstrate use of the Standard C Library. These applications reside in `AppPkg/Applications`.
Enquire	This is a program that determines many properties of the C compiler and the target machine that Enquire is run on. The only changes required to port this 1990s era Unix program to EDK II were the addition of 8 pragmas to enquire.c in order to disable some Microsoft VC++ specific warnings.
Hello	This is a very simple EDK II native application that doesn't use any features of the *Standard C Library*.
Main	This application is functionally identical to Hello, except that it uses the *Standard C Library* to provide a main() entry point.
Python	A port of the Python-2.7.1 interpreter for UEFI. This application is disabled by default. Un-comment the line for PythonCore.inf in the [Components] section of AppPkg.dsc to enable building Python.
StdLib	The StdLib package contains the standard header files as well as implementations of the standard libraries. Currently, the only standard library provided is the *Standard C Library*, the implementation of which is in the `StdLib/LibC` directory.
StdLibPrivateInternalFiles	The contents of this package are for the exclusive use of the library implementations in StdLib. Please do not use anything from this package in your application or unexpected behavior may occur. This package may be removed from a future release.

RELEASE NOTES

This Alpha release of the EADK has some restrictions, as described below.

1. Only the Microsoft VS2005 and VS2008, Intel C Compiler 10.1 (or later), GCC 4.3 (mingw32), GCC 4.4, and GCC 4.5 C compilers are supported for Ia32 or X64 CPU architectures.
2. The target machine must be running EDK II based firmware with the EDK II HII present and enabled.
3. The EADK has not been through Intel's Quality Assurance process. This means that specified standards compliance has not been validated, nor has it undergone formal functionality testing.
4. Applications must be launched from within the EFI Shell.
5. All file paths must use the forward slash, '/', as the separator character.
6. Absolute file paths may optionally be prefixed by a volume specifier such as "FS0:". The volume specifier is separated from the remainder of the path by a single colon '**:**'. The volume specifier must be one of the Shell's mapped volume names as shown by the "map" command.
7. Absolute file paths that don't begin with a volume specifier; e.g. paths that begin with "**/**", are relative to the currently selected volume.
8. The tmpfile(), and related, functions require that the current volume have a temporary directory as specified in <paths.h>. Currently, this is "`/Efi/Temp`".
9. There is a known issue with Console input hanging. Regular file I/O works fine.

The Standard C Library provided by this package is a "hosted" implementation conforming to the ISO/IEC 9899-1990 C Language Standard with Addendum 1. This is commonly referred to as the "C 95" specification.

The following instructions assume that you have an existing EDK II or UDK 2010 source tree that has been configured to build with your tool chain. For convenience, it is assumed that your EDK II source tree is located at `C:\Source\Edk2`.

INSTALLATION

The EADK is integrated within the EDK II source tree and is included with current EDK II check-outs. If they are missing from your tree, they may be installed by extracting, downloading or copying them to the root of your EDK II source tree. The three package directories should be peers to the Conf, MdePkg, Nt32Pkg, etc. directories.

The Python 2.7.1 distribution must be downloaded from python.org before the Python application can be built. Extracting Python-2.7.1.tgz into the AppPkg\Applications\Python directory will produce a Python-2.7.1 directory containing the Python distribution. Python files that had to be modified for EDK II are in the AppPkg\Applications\Python\PyMod-2.7.1 directory. These files need to be copied into the corresponding directories within Python-2.7.1.

There are some boiler-plate declarations and definitions that need to be copied into your application's INF and DSC build files. These are described in the **CONFIGURATION** section, below.

BUILDING

It is not necessary to build the libraries separately from the target application(s). If the application references the libraries, as described in **USAGE**, below; the required libraries will be built as needed.

To build the applications included in AppPkg, one would execute the following commands within the "Visual Studio Command Prompt" window:

```
> cd C:\Source\Edk2
> .\edksetup.bat
> build –a X64 –p AppPkg\AppPkg.dsc
```

This will produce the application executables: Enquire.efi, Hello.efi, and Main.efi in the `C:\Source\Edk2\Build\AppPkg\DEBUG_VS2008\X64` directory; with the DEBUG_VS2008 component being replaced with the actual tool chain and build type you have selected in Conf\Tools_def.txt. These executables can now be loaded onto the target platform and executed.

If you examine the AppPkg.dsc file, you will notice that the StdLib package is referenced in order to resolve the library classes comprising the *Standard C Library*. This, plus referencing the StdLib package in your application's .inf file is all that is needed to link your application to the standard libraries.

USAGE

This implementation of the Standard C Library is comprised of 16 separate libraries in addition to the standard header files. Nine of the libraries are associated with use of one of the standard headers; thus, if the header is used in an application, it must be linked with the associated library. Three libraries are used to provide the Console and File-system device abstractions. The libraries and associated header files are described in the following table.

Library Class	Header File(s)	Notes
LibC	-- Use Always --	This library is always required.
LibCtype	ctype.h, wctype.h	Character classification and mapping
LibLocale	locale.h	Localization types, macros, and functions
LibMath	math.h	Mathematical functions, types, and macros
LibStdio	stdio.h	Standard Input and Output functions, types, and macros
LibStdLib	stdlib.h	General Utilities for numeric conversion, random num., etc.
LibString	string.h	String copying, concatenation, comparison, & search
LibSignal	signal.h	Functions and types for handling run-time conditions
LibTime	time.h	Time and Date types, macros, and functions
LibUefi	sys/EfiSysCall.h	Provides the UEFI system interface and "System Calls"
LibWchar	wchar.h	Extended multibyte and wide character utilities
LibNetUtil		Network address and number manipulation utilities
DevConsole	Automatically provided	File I/O abstractions for the UEFI Console device. No need to list this library class in your INF file(s).

Library Class	Header File(s)	Notes
DevShell	Add if desired	File I/O abstractions using UEFI shell facilities. Add this to the application's main INF file if file-system access needed.
DevUtility	-- Do Not Use --	Utility functions used by the Device abstractions
LibGdtoa	-- Do Not Use --	This library is used internally and should not need to be explicitly specified by an application. It must be defined as one of the available library classes in the application's DSC file.

Table 1: Standard Libraries

These libraries must be fully described in the [LibraryClasses] section of the application package's DSC file. Then, each individual application needs to specify which libraries to link to by specifying the Library Class, from the above table, in the [LibraryClasses] section of the application's INF file. The AppPkg.dsc, StdLib.dsc, and Enquire.inf files provide good examples of this. More details are in the **CONFIGURATION** section, below.

Within the source files of the application, use of the Standard headers and library functions follow standard C programming practices as formalized by ISO/IEC 9899:1990, with Addendum 1, (C 95) C language specification.

CONFIGURATION

DSC Files

All EDK II packages which build applications that use the standard libraries must include some "boilerplate" text in the package's .dsc file. The text can be copied from either AppPkg.dsc or StdLib.dsc. Each affected section of the DSC file is described below.

```
[LibraryClasses]
  #
  # Common Libraries
  #
  BaseLib|MdePkg/Library/BaseLib/BaseLib.inf
  BaseMemoryLib|MdePkg/Library/BaseMemoryLib/BaseMemoryLib.inf

  TimerLib|PerformancePkg/Library/DxeTscTimerLib/DxeTscTimerLib.inf
  # To run in an emulation environment, such as NT32, comment out
  # the TimerLib description above and un-comment the line below.
  # TimerLib| MdePkg/Library/BaseTimerLibNullTemplate/BaseTimerLibNullTemplate.inf

  #
  # C Standard Libraries
  #
  LibC|StdLib/LibC/LibC.inf
  LibStdLib|StdLib/LibC/StdLib/StdLib.inf
  LibString|StdLib/LibC/String/String.inf
  LibWchar|StdLib/LibC/Wchar/Wchar.inf
  LibCType|StdLib/LibC/Ctype/Ctype.inf
  LibTime|StdLib/LibC/Time/Time.inf
  LibStdio|StdLib/LibC/Stdio/Stdio.inf
  LibGdtoa|StdLib/LibC/gdtoa/gdtoa.inf
  LibLocale|StdLib/LibC/Locale/Locale.inf
  LibUefi|StdLib/LibC/Uefi/Uefi.inf
  LibMath|StdLib/LibC/Math/Math.inf
  LibSignal|StdLib/LibC/Signal/Signal.inf
  LibNetUtil|StdLib/LibC/LibGcc/LibGcc.inf
```

```
        # Libraries for device abstractions within the Standard C Library.
        # Applications should not directly access any functions defined
        # in these libraries.
          DevUtility|StdLib/LibC/Uefi/Devices/daUtility.inf
          DevConsole|StdLib/LibC/Uefi/Devices/daConsole.inf
          DevShell|StdLib/LibC/Uefi/Devices/daShell.inf
```
Figure 1: Library Class Descriptions

Descriptions of the Library Classes comprising the Standard Libraries must be included in your application package's DSC file, as shown in Figure 1, above.

The NULL TimerLib instance must be selected if you desire to run your application under an emulation environment – unless there is a supported TimerLib for that environment. For example, the InOsEmuPkg provides a DxeTimerLib which can be used for the TimerLib instance. If that is the case, the changes described for emulation environments, in Figure 3: Package Build Options, can be skipped.

```
        [Components]
        # BaseLib and BaseMemoryLib need to be built with the /GL- switch
        # when using the Microsoft tool chains.  This is required so that
        # the library functions can be resolved during the second pass of
        # the linker during link-time-code-generation.
        #
          MdePkg/Library/BaseLib/BaseLib.inf {
            <BuildOptions>
              MSFT:*_*_*_CC_FLAGS = /X /Zc:wchar_t /GL-
          }
          MdePkg/Library/BaseMemoryLib/BaseMemoryLib.inf {
            <BuildOptions>
              MSFT:*_*_*_CC_FLAGS = /X /Zc:wchar_t /GL-
          }
```
Figure 2: Package Component Descriptions

The directives in Figure 2 will create instances of the BaseLib and BaseMemoryLib library classes that are built with Link-time-Code-Generation disabled. This is necessary when using the Microsoft tool chains in order to allow the library's functions to be resolved during the second pass of the linker during Link-Time-Code-Generation of the application.

```
        [BuildOptions]
          INTEL:*_*_IA32_CC_FLAGS  = /Qfreestanding
           MSFT:*_*_IA32_CC_FLAGS  = /X /Zc:wchar_t
            GCC:*_*_IA32_CC_FLAGS  = /ffreestanding –nostdinc –nostdlib

        # The Build Options, below, are only used when building the C library
        # to be run under an emulation environment.  The clock() system call
        # is modified to return -1 indicating that it is unsupported.
        # Just un-comment the lines below and select the correct
        # TimerLib instance, above.

          # INTEL:*_*_IA32_CC_FLAGS  = /D NT32dvm
          #  MSFT:*_*_IA32_CC_FLAGS  = /D NT32dvm
          #   GCC:*_*_IA32_CC_FLAGS  = -DNT32dvm
```
Figure 3: Package Build Options

Each compiler assumes, by default, that it will be used with standard libraries and headers provided by the compiler vendor. Many of these assumptions are incorrect for the UEFI environment. By including a BuildOptions section, as shown in Figure 3, these assumptions can be tailored for compatibility with UEFI and the EDK II Standard Libraries.

INF Files

The INF files for most modules will not require special directives in order to support the Standard Libraries. The two cases which could occur are described below.

```
        [LibraryClasses]
          UefiLib
          LibC
          LibString
          LibStdio
          DevShell
```

Figure 4: Module Library Classes

Modules of type UEFI_APPLICATION that perform file I/O should include library class DevShell. Including this library class will allow file operations to be handled by the UEFI Shell. Without this class, only Console I/O is permitted.

```
         [BuildOptions]
          INTEL:*_*_*_CC_FLAGS          = /Qdiag-disable:181,186
           MSFT:*_*_*_CC_FLAGS          = /Oi- /wd4018 /wd4131
            GCC:*_*_IPF_SYMRENAME_FLAGS = --redefine-syms=Rename.txt
```

Figure 5: Module Build Options

An application's INF file may need to include a [BuildOptions] section specifying additional compiler and linker flags necessary to allow the application to be built. Usually, this section is not needed. When building code from external sources, though, it may be necessary to disable some warnings or enable/disable some compiler features.

IMPLEMENTATION-Specific Features

It is very strongly recommended that applications not use the **long** or **unsigned long** types. The size of this type varies between compilers and is one of the less portable aspects of C. Instead, one should use the UEFI defined types whenever possible. Use of these types, listed below for reference, ensures that the declared objects have unambiguous, explicitly declared, sizes and characteristics.

UINT64	**INT64**	**UINT32**	**INT32**	**UINT16**	**CHAR16**
INT16	**BOOLEAN**	**UINT8**	**CHAR8**	**INT8**	
UINTN	**INTN**			**PHYSICALADDRESS**	

There are similar types declared in sys/types.h and related files.

The types UINTN and INTN have the native width of the target processor architecture. Thus, INTN on IA32 has a width of 32 bits while INTN on X64 and IPF has a width of 64 bits.

For maximum portability, data objects intended to hold addresses should be declared with type intptr_t or uintptr_t. These types, declared in sys/stdint.h, can be used to create objects capable of holding pointers. Note that these types will generate different sized objects on different processor architectures. If a constant size across all processors and compilers is needed, use type PHYSICAL_ADDRESS.

Though not specifically required by the ISO/IEC 9899 standard, this implementation of the *Standard C Library* provides the following system calls which are declared in `sys/EfiSysCall.h`.

close	dup	dup2	fcntl
fstat	getcwd	ioctl	isatty
lseek	lstat	mkdir	open
poll	read	rename	rmdir
stat	unlink	write	

The open function will accept file names of "stdin:", "stdout:", and "stderr:" which cause the respective streams specified in the UEFI System Table to be opened. Normally, these are associated with the console device. When the application is first started, these streams are automatically opened on File Descriptors 0, 1, and 2 respectively.